Recursion - 1

Doubt →

hello how are you

← ollo wok era woy

[→ Fermat's Thm]

Doubt Session

Recursion - 1

Technique

Problem → Small Problems (Assume Solution exists)

↑ Recursively

n ← $n-1$

← $(n-1)+(n-2)$

← $n/2$ $n/2$

Assume

Rule

1. Find out the solution for smallest N. [Base Case]
2. Assume Subproblem can be solved
3. Express $f(n)$ in terms of smaller subproblem $f(n')$ [Rec Case] [Rec. Case] $n' < n$

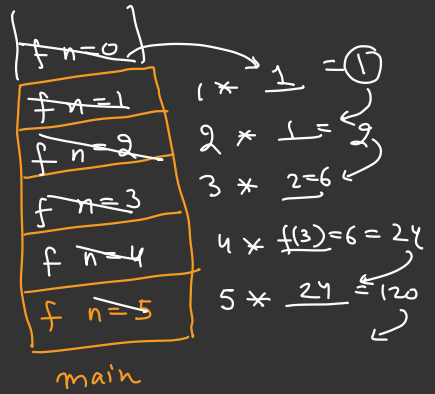
Factorial

$\Rightarrow 5! = 5 * 4!$

- $\Rightarrow f(n) = n * f(n-1)$ Rec Case
- $\Rightarrow f(0) = 1$

```
int f(int n){
    // Base Case
    if(n==0)
        return 1
    // Rec Case
    return n * f(n-1)
}
```

f(5)



$N * 1 = O(N)$

- Time → Calls * Work in each call
- Space → Max Stack Depth * Space in each stack frame

$= N \times 1$

$= O(N)$

(8) Given an array of size N, Check if the array is sorted [Recursion]


```
bool isSorted( A , N ) {
    if(N==1 or N==0){
        return true;
    }
    A' = subarray with 0 element removed!   // new int[n-1], O(N)
    return A[0]<A[1] && isSorted(A', N-1);
}
```

new array with 0th element removed

Time = N × N = $O(N^2)$ Bad :(

A: 1 3 5 7 12

⇒ 1<3 && isSorted(3,5,7,12)

3<5 && isSorted(5,7,12)

5<7 && isSorted(7,12)

7<12 && isSorted(12)

N calls

⇒ 3 5 9 7

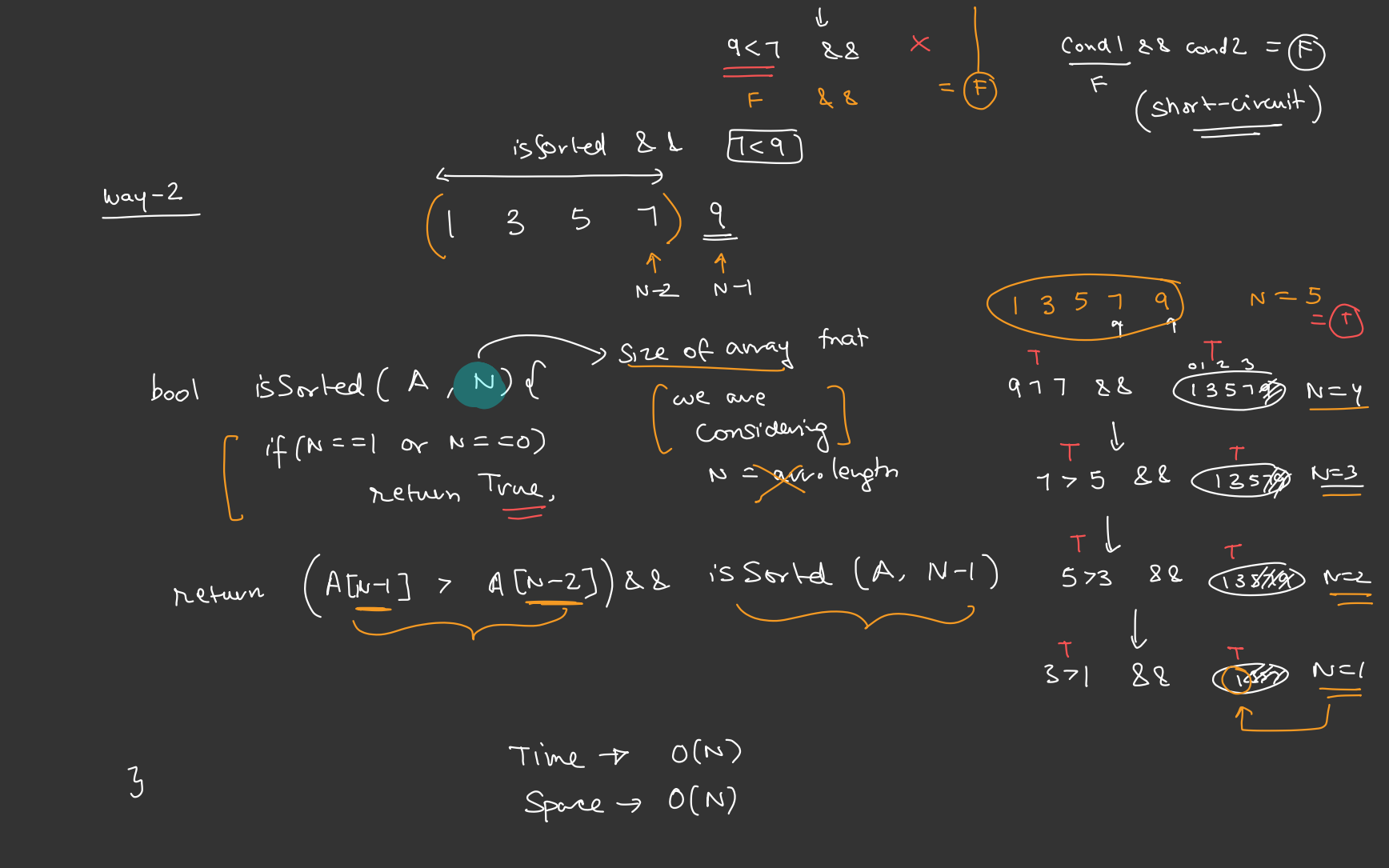
3<5 && isSorted(5,9,7)

5<9 && isSorted(9,7)

isSorted(9,7) = F

isSorted(5,9,7) = F

Main = F


9<7 && = F
F && = F
cond1 && cond2 = F
F
(short-circuit)
isSorted && 7<9
1 3 5 7 9
N-2
N-1
Way-2
bool isSorted (A , N) {
if (N == 1 or N == 0)
return True,
Size of array that we are considering
N = arr.length
return (A[N-1] > A[N-2]) && isSorted (A, N-1)
1 3 5 7 9
N = 5 = T
9 > 7 && 1 3 5 7 9 N=4
7 > 5 && 1 3 5 7 N=3
5 > 3 && 1 3 5 N=2
3 > 1 && 1 3 N=1
}
Time → O(N)
Space → O(N)

Space → O(N)

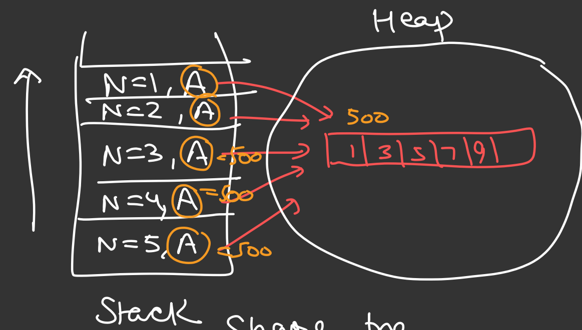
```
f(A, N) {          // A = 500

  → f(A, N-1);     // A = 500, N-1: how many element to consider

}

main() {
    A = new int[5]

}
```

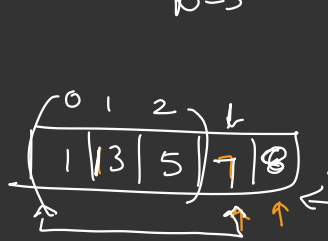


Share the same array

Function calls have a copy of array reference variable

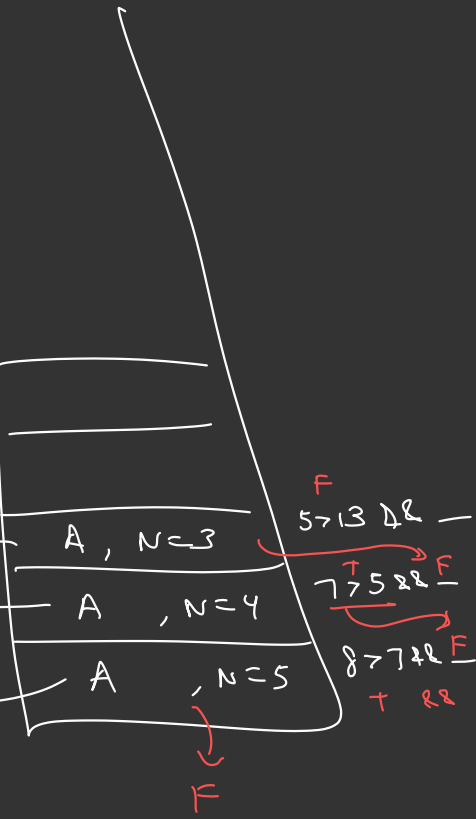
```
bool isSorted ( A , N ) {
    [ if (N == 1 or N == 0)
          return True,

    return (A[N-1] > A[N-2]) && isSorted (A, N-1)
}
```



N=3

N



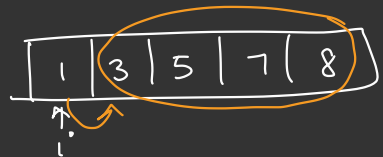
5<3 && True = false

↑ false

(6 <= 2)

↓ bool

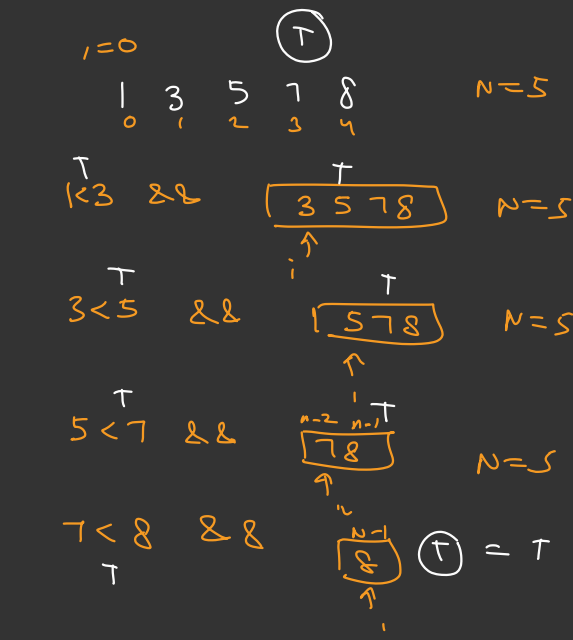
Way - 3

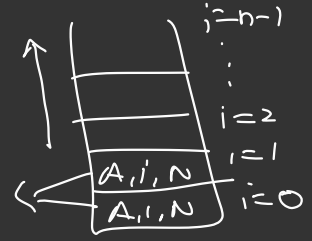



```
isSorted (A, i, N){

    if( i == n-1 ){
        return true;
    }

    return A[i] < A[i+1] &&
           isSorted(A, i+1, N);
}
```





TC = No of calls * work

= N * 1 = O(N)

Space = O(N)

Rec → Many algorithms are very hard to write iteratively

→ Backtracking based problems.

→ Teaches you "how to think"

↳ Rec Code → Itr Code

Simple challenge

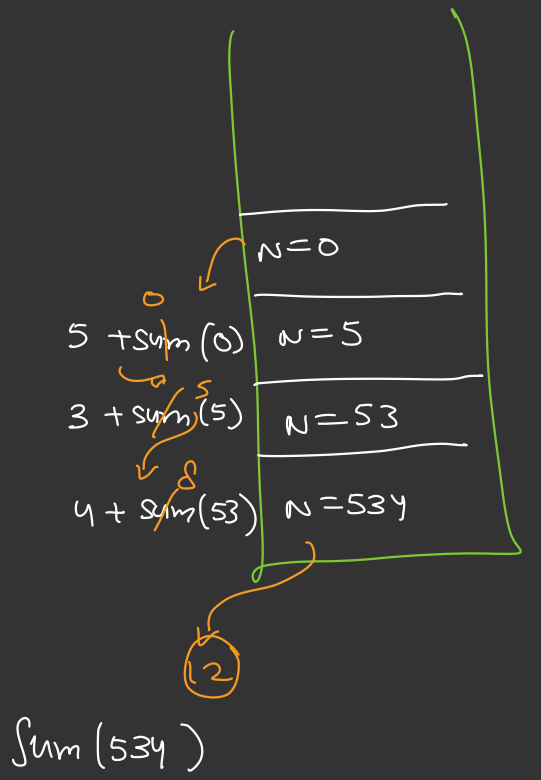
Given N, find sum of digits of N [Rec.]

```
int Sum (int N) {
Base [ if (N==0)
         return 0
Rec  [ return N%10 + Sum(N/10),
}
```

Time ⇔ Space

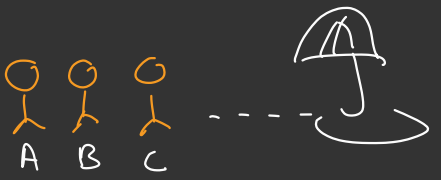
↳ O(No of Digits in N)

Sum(N) = N%10 + Sum(N/10)

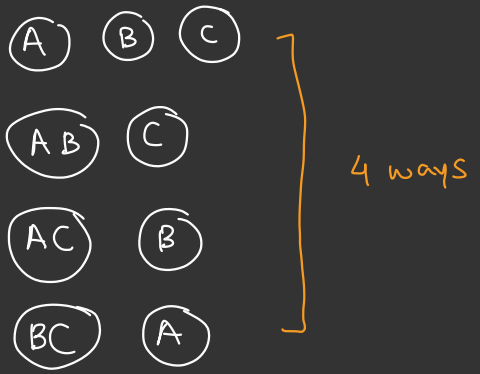


CHALLENGE

Q. N friends want to go to party
Each friend can go solo or as a couple
Find no of ways in which they can go.



N = 3

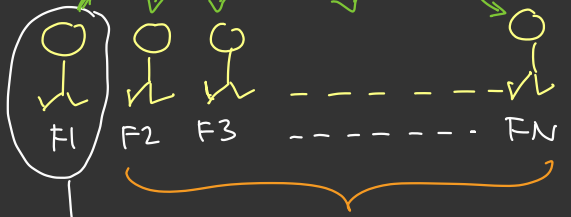


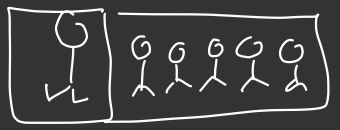
$A_1, A_2 \text{ ---- } A_n$

$f(N) = ?$

Break it, Make it

→ Solve for one friend, let rec. solve for Rest





Choices

F1 and

Solo × other ppl

OR

Pair × other ppl

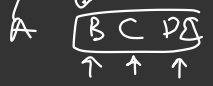
$$f(N) = 1 \times f(n-1) \quad \text{OR} \quad (N-1) * f(N-2)$$

$$f(N) = f(n-1) + (N-1)\, f(N-2)$$

Base Case

$$f(1) = 1$$
$$f(2) = 2$$

$^{N-1}C_1$



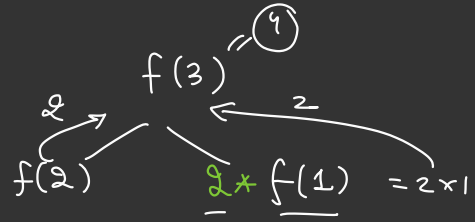
$^{3}C_1 = 3$

AB CD E

f(2) A, B

f(1) (AB)

```
int ways(int n){
    if(n<=2){
        return n;
    }
    return ways(n-1) + (n-1)ways(n-2);
}
```



$f(3) = 2 + 2*1 = 4$

$f(4) = f(3) + 3 * f(2)$

$= 4 + 3 * (2) = 10$ ways

A B C D

A Solo

A Pair

(A), B, C, D — 1

AB, C, D — 2

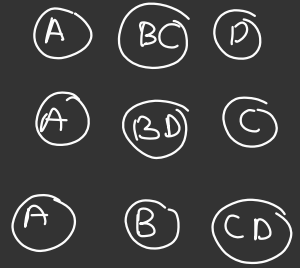
AB, CD — 2

AC, B, D — 2

AC, BD — 2

AD, B, C — 2

AD, BC — 2

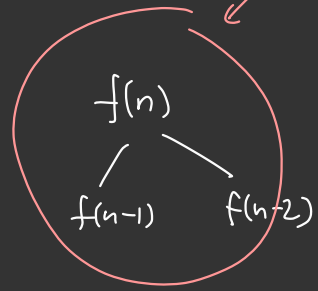


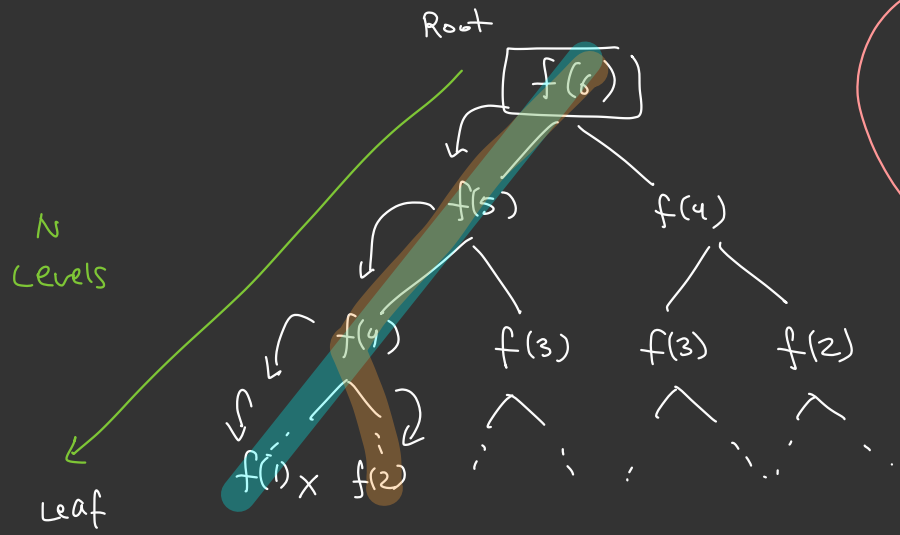
3x2

= 6

4+6 = 10

just like fibonacci





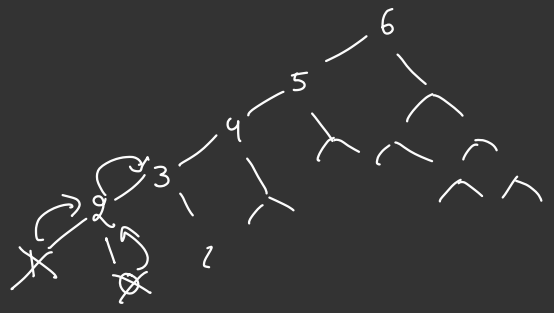
$1 + 2 + 4 + \ldots\ldots 2^{n-1}$

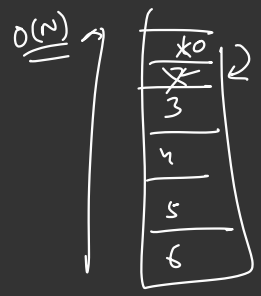
$= \frac{2^n - 1}{2 - 1}$

Time = $O(2^n)$ fn calls

Space = Max Depth

↳ = $O(N)$





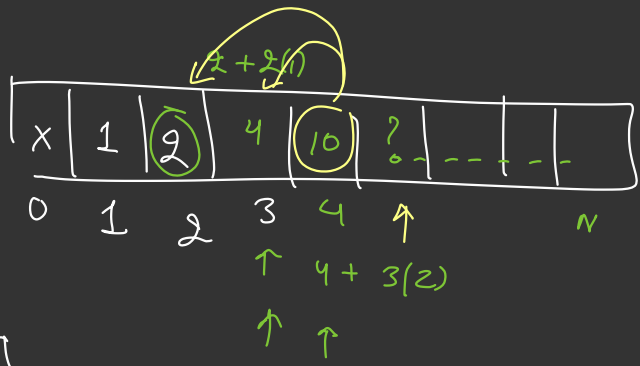
Rec. Thinking

↓

Directly ?

Iterative Code

```
for( i=1 ---- N){
    arr[i] = arr[i-1] + (i-1) * arr[i-2]
}
```



Current Term = 3 variables

↓ ↓ ↑

Prev — Prev prev

O(1) space

⇒ O(N) time

⇒ O(N) space

↑

??

3 variables

10:40 <- Break -->

Indian Money Change Coins = [1, 2, 5, 10, 20, 50, 100, 200, 500, 2000]

Constant size

276

Given N Rupees, min notes/coins needed make a change for N.

N = 276 - 200
= 76 - 50
= 26 - 20
= 6 - 5
= 1 - 1
= 0 Stop

200 + 50 + 20 + 5 + 1

"Greedy Choice"

⇒ Pick the biggest note ≤ N everytime

Search → Linear Search → Anyone. O(1)
Search → Binary Search → Anyone. O(1)

Base Case $f(0) = 0$

Rec Case $f(N) = 1 + f(N - \text{largest note})$

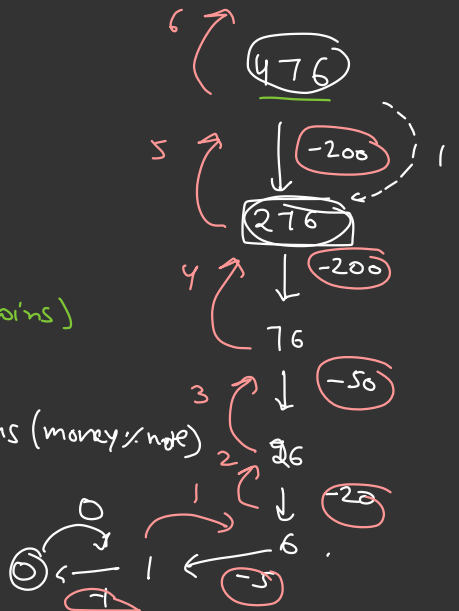
min coins / notes

$276 \rightarrow$ (1) Note 200 + f(76) 4

$= 5$ Notes

Coins = [1, 2, 5, 10, ...]

```
int minCoins(money, coins){
    if(money == 0){
        return 0;
    }
    note = search(money, coins)        O(1)
    print(note, money/note)
    return money/note + minCoins(money % note)
}
```



money >>> 2000

₹ 1 million

1000000 - 2000

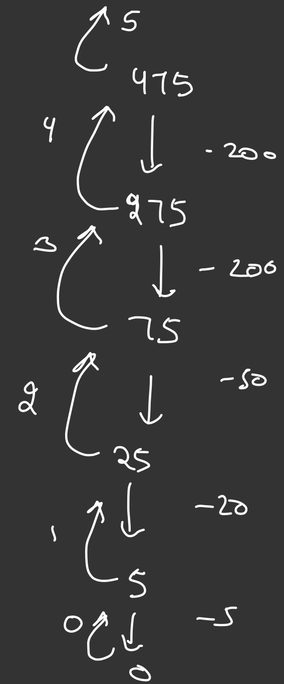
= 9999958000

10,00 20

→ $\left(\frac{10{,}020}{2000}\right) + f(20)$

coinss

5 + f(20)



$$f(\text{money}) = \frac{\text{money}}{\text{note}} + f(\text{money} \% \text{note})$$

atmax 1 for every type note/coins

$2 \times 2000 + 2 \times 500 = 5000$

$\left(\frac{5000}{2000}\right) + f(1000)$

$= 2 + f(1000) = 4$

$f\left(\frac{1000}{500}\right) + f(0) = 2 + 0 = 2$

$5000 = 2 \times 2000 + 2 \times 500$

time $\rightarrow$ $O(\text{len(coins)})$

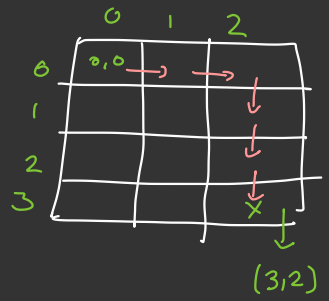
$= O(1)$

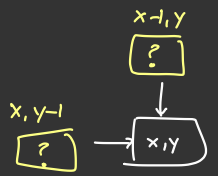
$567 =$ _ $* c1$
_ $* c2$
_ $* c3$
⋮
_ $* 4$

[1, 2, 5, 10, 20 . . . . 2000]

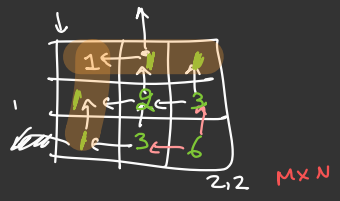
2D grid problem

No of ways to reach (3,2)





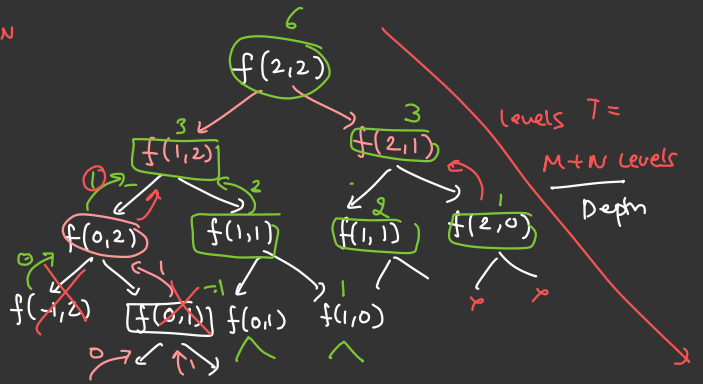
Rec Case • $f(x,y) = f(x-1,y) + f(x,y-1)$



Base Case $f(0,0) = 1$

```
int f(x,y){
    if(x==0 || y==0)
        return 1;

    return f(x-1,y)
    +f(x,y-1)
}
```



3

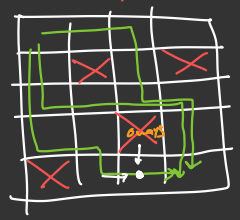
Time = 2^{T}

$= 2^{M+N}$

Space = $O(M+N)$

Twist

char array 2D array



```
int f(x,y){
  if(x,y == 0,0) {return 1}
  if(x,y) is blocked or outside
      return 0,
  return f(x-1,y) + f(x,y-1);
}
```

```
arr[x][y] == 'X'
  return 0;
```

⇒ [Doubt]

⇒ "Hello _ World _ X _ Y

olleH dlrow _ _

Java / Python

→ split()

String = [" olleH ", _]

Join

```
reverseWord( word, s, e ) {
    if(s >= e) { return }
    swap(a[s], a[e]);
    reverseWord(word, s+1, e-1)
}
```